International Journal of Hybrid Information Technology
Vol.7, No.3 (2014), pp.363-376
http://dx.doi.org/10.14257/ijhit.2014.7.3.34

Preserving Privacy of Cloud Data Using Homomorphic Encryption in MapReduce

Sujitha . G, Rajeswaran .T, Thiagarajan.R ,Vidya. K Mercy Shalinie. S
Anna University, Chennai, TamilNadu, India
sujitha@auist.net

Abstract

In recent years, outsourcing large amount of data in cloud and how to manage the data raises many challenges with respect to privacy. The concerns of privacy can be addressed if cloud users encrypt the data deployed in the cloud. Among the various cryptographic encryption schemes, homomorphic scheme allow to perform meaningful computations on encrypted data. In this context, the research deals with homomorphic encryption scheme for maintaining privacy and security in cloud by detecting the error incurred while transferring data using RSA cryptosystem. Three types omomorphic error detection schemes proved that preserving privacy seems to be efficient using Map-Reduce Model.

Keywords: *Cloud Security, Data Privacy, Homomorphic Encyrption, MapReduce*

1. Introduction

Cloud computing is started off with Grid Computing, where large number of systems are used for solving scientific problem that require high levels of parallel computation. This technology expanded exceptionally, which eventually stimulated concerns over ensuring data security in public networks

According to a recent Survey conducted by Cisco Global Cloud Networking Academy, it has been revealed that 72 percent of IT professionals stated that security of data is a major hindrance to implement the services in cloud [1]. Recent development in cloud storage and the services rendered by it allows users to outsource storage. As a result, it allows companies or organizations to offload the task of maintaining datacenters. In the past few years, the security requirements for data are very strong and many algorithms have evolved [2]. Only a few algorithms play a comprehensive role in creating and maintaining a secure session over vulnerable public networks. Public key cryptography is one of the commonly used algorithms for this type of operation. The authenticity between the communicating parties is ensured by implementing this technique. These communicating parties share their private keys amongst them before exchanging information. In the case of transmitting a message over a public channel, the work of Diffie Helman [1] and RSA [3] provides way to encrypt a message into cipher text using private key. Consequently, the receiver on the other side has to read the cipher text by decryption with the help of their private key. The encryption scheme shows that the secret decryption key allows retrieving the actual text but if the secret key is lost, the ciphertext is of no use. In 1978, RDA [4] decided to propose a technique on performing arbitrary computations on encrypted data. Such techniques give rise to useful applications to privately perform manipulations on encrypted data in the cloud. The necessary data can be decrypted by performing their corresponding computations. Assuring privacy tend to be very critical when complex computations are performed on encrypted data.

ISSN: 1738-9968 IJHIT

Homomorphic encryption is evolved to solve such critical issues. The homomorphic properties of ciphers have been implemented in various real time applications. Some of them include privacy protection during electronic voting, computation in multiparty environment computation and analyzing traffic in distributed environment [3]. Basically, homomorphic encryption enhances the security measures of cloud data. Data protection is achieved through the homomorphic encryption scheme, which allows additive and multiplicative operations over encrypted bits. The cloud provider accepts encrypted user query data to perform processing without being aware of its content. The results of the user query which is again an encrypted data is sent to the user. The user alone decrypts the data and views the result of the query.

The public-key and private-key cryptosystems are designed with various fault attacks [5]. Error Detection (ED)-based countermeasures have been developed for both private-key cryptosystems such as AES, public-key cryptosystems such as RSA[6], ECC [7][8]. In this research, the focus is on detecting the fault attack using public-key cryptography, RSA. It is identified that countermeasures for RSA can be devised. It is achieved through the digital signature mode which is based on CRT-RSA.

In the past years, homomorphic Encryption allows simple computation on encrypted data. Such practice is known for a long time. The GM [9] encryption scheme supports addition of encrypted bits mod 2 (Exclusive OR function). A number of encryption systems that are either additively or multiplicatively homomorphic followed the suit. Encryption systems of ElGamal encryption scheme [10], The Paillier encryption scheme[11][12] and its generalization [2], a host of lattice-based encryption schemes and others evolved [13][14][15]. A system used in involved additive and multiplicative encrypted texts which has more number of additions and just one multiplication. Constructing an encryption scheme that is both additively and multiplicatively homomorphic remained a major challenge [16]. The additive and multiplicative homomorphisms form a complete set of operations. The scheme enables performing any polynomial-time computation on encrypted data. Later, Gentry[16] constructed a fully homomorphic encryption which allows evaluation of arbitrary number of additions and multiplications on encrypted data [17][18]. Rivest et al [19] Gentry and Halevi[16], Diffie and Hellman[20] proposed the RSA with multiplicative and additive homomorphism respectively.

2. Fault Attack on Cryptographic Implementations

Cryptographic algorithms like symmetric ciphers, asymmetric ciphers, and hash functions are designed with a set of primitives that meet specific objectives (Guan et al 2013). The Cryptographic implementations on evaluation show their resistivity against attacks. It is necessary to determine countermeasures against such attacks and evaluate the feasibility and applicability of such attacks. Side channel attacks assist in breaking the hardware or software implementations of many cryptosystems including block ciphers (DES, AES), stream ciphers (RC4, RC6), public key ciphers (RSA-type ciphers, ElGamal-type ciphers, ECC, XTR, etc.), to break the implementations of signature schemes, chunk authentication code schemes, cryptographic protocols, cryptosystems, and networking systems. Side channel faults are of two kinds. The first kind of fault is induced during cryptographic computation. These faults are either random or intentional, caused by a voltage manipulation. The second kinds of faults occur by intentionally injecting corrupted input data. This research focuses on such computation wherein the receiver while noticing a mismatch identifies that the chunk is faulty.

3. Fault attack on RSA

There are many formal definitions for public key cryptosystems such as RSA and Pailler cryptosystem. Public-key cryptography is asymmetric since one of the participants has a secret key, while the others have access to the public key that matches the secret key. But, the symmetric system has only one key which should be shared between the two participants. The complexity of the systems indicates that the computation of public key systems is time consuming. The objective is to exchange data between two users without sharing a common secret.RSA Labs embarked on an effort to differentiate the security level of symmetric key and the RSA key size (Yu et al 2013b). The security of RSA depends on the key size. In integer factorization problem-based algorithms, the security depends on the difficulty to factorize a large number to obtain large primes.

3.1. RSA Encryption Scheme

A public-key encryption scheme E1 is a tuple which is represented as (Enc; Dec, KeyGen). The key generation algorithm (Enc, Dec, KeyGen) takes the security parameter k1 as input and outputs a pair of keys (pubk; seck). pubk refers to public key and seck refers to private key or secret key. pubk and seck each have length at least k1, and that k1can be determined from pubk; seck.

(1) Key Generation: Choose two distinct prime numbers p1 and q1. For security purposes, the integers p1 and q1 should be chosen at random, and should be of similar bit length. Prime integers can be efficiently found using a primality test.

- Compute n1 = p1*q1. n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
- Compute $\varphi(n1) = \varphi(p1)\varphi(q1) = (p1 - 1)(q1 - 1)$, where φ is Euler's totient function.
- Choose an integer e1 such that $1 < e1 < \varphi(n)$ and gcd $(e1, \varphi(n1)) = 1$; i.e e1 and $\varphi(n1)$ are co-prime. e1 is released as the public key exponent. e1 having a short bit-length.
- Determine the multiplicative inverse of e1 (modulo $\varphi(n1)$ as $d1-1 \equiv e1 \pmod{\varphi(n1)}$, i.e., d1 is the multiplicative inverse of e1). This is more clearly stated as: solve for d given $d1 \cdot e1 \equiv 1 \pmod{\varphi(n1)}$. This is computed using the extended Euler function. d1 is kept as the private key exponent. The public key consists of the modulus n1 and the public (or encryption) exponent e1. The private key consists of the modulus n1 and the private (or decryption) exponent d1, which must be kept secret. p1, q1, and $\varphi(n1)$ must also be kept secret because they can be used to calculate d1.

(2) Encryption process is a technique to convert the plaintext into ciphertext. Public key is used for encryption process. Public key pair is given as (*n, e).* Message encryption process is done using the value of (n,e) and is represented as :

$$C=M^e(\text{mod } n) \quad (1)$$

M denotes the input message. C denotes ciphertext.

Encryption algorithm (Enc) uses a public-key pubk and a string m1 called the message from some underlying message space (M1) as input. It produces a ciphertext c1 from an underlying ciphertext space (C1).

(3) Decryption process is a technique to convert the ciphertext into plaintext. Private key is used for decryption process. Decryption algorithm uses a private-key pair(n1,e1) and a ciphertext C1 as input and produces an output message M1.

It is represented as:

$$M=C^e(\text{mod } n) \quad (2)$$

M denotes the output message. C denotes the ciphertext messages.

3.2. RSA Homomorphic Property

A partially or fully homomorphic system requires another algorithm to perform operations on ciphertexts(He et al 2012). The public-key encryption scheme E = (Enc, Dec, KeyGen) is homomorphic if for all k and all (pk; sk) output from KeyGen(k), it is possible to define groups M,C so that: The plaintext space M, and all ciphertexts output by Enc_{pk} are elements of C. For any m1, m2€M and c1, c2 € C with m1 = Dec_{sk} (c1) and m2 = Dec_{sk} (c2) it holds that: Dec_{sk} (c1 *c2) = m1 * m2 where the group operations * are carried out in C and M, respectively. and c1, c2 € C1 with m1 = Dec seck (c1) and m2 = Dec seck (c2) it holds that: Dec seck(c1 *c2) = m1 * m2.

3.3. Paillier Encryption Scheme

As discucced above, public-key encryption scheme tuple E is represented as (Enc, Dec, KeyGen)

(1) Key Generation: The key generation algorithm is similar to RSA which follows the steps as:

- Choosing two large prime numbers *p1* and *q1* randomly and independently of each other such that gcd(p1q1,(p1-1) (q1-1))=1. This property is assured if both primes are of equivalent length, i.e., $p1, q1\epsilon 1||\{0,1\}^{s\varepsilon-1}$ for security parameter.
- Computation of n1=p1*q1 and $\lambda 1 = lcm(p1-1, q1-1)$.
- Selecting random integer g1 where $g1\varepsilon Z_{n^2}^*$ which ensure that n1divides the order of g1 by checking the existence of the following modular multiplicative inverse:

$$\mu 1 = (L1(g1^{\lambda 1} \ mod \ n1^2))^{-1} \quad (3)$$

where function L1 is defined as $(u1) = \frac{u1-1}{n1}$.

Note that the notation $\frac{a1}{b1}$ does not denote the modular multiplication of a1 times the modular multiplicative inverse of b1 but rather the quotient of a1 divided by b1, i.e., the largest integer value v $1 \geq 0$ to satisfy the relation a$1 \geq v1b1$

- The public (encryption) key is (n1, g1)
- The private (decryption) key is ($\lambda 1$, $\mu 1$)
- If using p, q of equivalent length, a simpler variant of the above key generation steps would be to set $g1 = n + 1$, $\lambda 1 = \varphi(n1)$ and $\mu 1 = \phi(n1)^{-1}$, where $\phi(n1) = (p1 - 1)(q1 - 1)$.

(2) Encryption process is a technique to convert the plaintext into ciphertext. Encryption algorithm (Enc) uses a public-key pubk and a string m1 called the message from some underlying message space (M1) as input. It produces a ciphertext c1 from an underlying ciphertext space (C1).

- Let m be a message to be encrypted where m$1 \in Z_{m1}$
- Select random r1 where r$1 \epsilon Z_n^*$
- Compute cipher text as: c1=$g1^{m1}.r1^{n1} \; mod \; n1^2$.

Public key is used for encryption process. Public key pair is given as (*n1, g1).* Message encryption process is done using the value of (n1,g1) and is represented as :

$$c1 = g1^{m1}.r1^{n1} \; mod \; n1^2 \quad (4)$$

m denotes the input message. c1 denotes ciphertext. r1 denotes random number.

(3) Decryption process is a technique to convert the ciphertext into plaintext. Private key is used for decryption process. Let c1 be the cipher text to decrypt, where c $1 \epsilon Z_{n1^2}^*$. Decryption algorithm uses a private-key pair ($\lambda 1$, $\mu 1$) and a ciphertext c1 as input and produces an output message M1.

It is represented as:

$$M1 = l1 \; (c1^{\lambda 1} \bmod n^2).\mu 1 \bmod n1 \quad (5)$$

M1 denotes the output message. 11 denotes the ciphertext messages.

3.4. Paillier Homomorphic Property

The other algorithm to support the homomorphic property of pailler cryptosystem is explained. A public-key encryption scheme E = (Enc, Dec, KeyGen) is homomorphic if for all k and all (pk; sk) output from KeyGen(k), it is possible to define groups M,C so that: The plaintext space M1, and all ciphertexts output by Enc_{pk} are elements of C. For any m1, m2€M and c1, c2 € C with $m1 = Dec_{sk}(c1)$ and $m2 = Dec_{sk}(c2)$ it holds that: $Dec_{sk}(c1 * c2) = m1 + m2$ where the group operations + are carried out in C and M, respectively and c1,c2 € C1 with m1 = Dec seck (c1) and m2 = Dec seck (c2) it holds that: Dec seck(c1*c2) = m1+m2.The addition of two ciphertexts will be equal to encrypt to the sum of their corresponding plaintexts.

The homomorphic encryption on cloud store proposed in the research preserve the privacy of data encrypted. The large data tested using additive and multiplicative homomorphic property is a time consuming process. It is controlled by an efficient application of the process in parallel mode [22]. Hadoop's Map-Reduce discussed in the previous chapter seems to be an attractive cost effective solution for large scale data processing services like securing data in the cloud through block encryption in parallel mode [23].

4. Homomorphic Based Error Detection Scheme

The error detection scheme includes input block that contains all the input chunks. Based on the size of the input, chunks are created. The number of chunks decides the type of error detection scheme. When the count of chunks is two, the basic error detection scheme for even chunks is evaluated as discussed above. When the count of chunks is odd the error detection scheme for odd chunks is evaluated. Under this scheme a constant chunk is generated and it is used during encryption. The large dataset involves large number of chunks relatively of the order of n follow the enhancement error detection scheme. The enhanced error detection scheme is allowed to run in parallel framework. K denotes number of input chunks taken for all three schemes. Figure 1 Homomorphic error detection scheme.

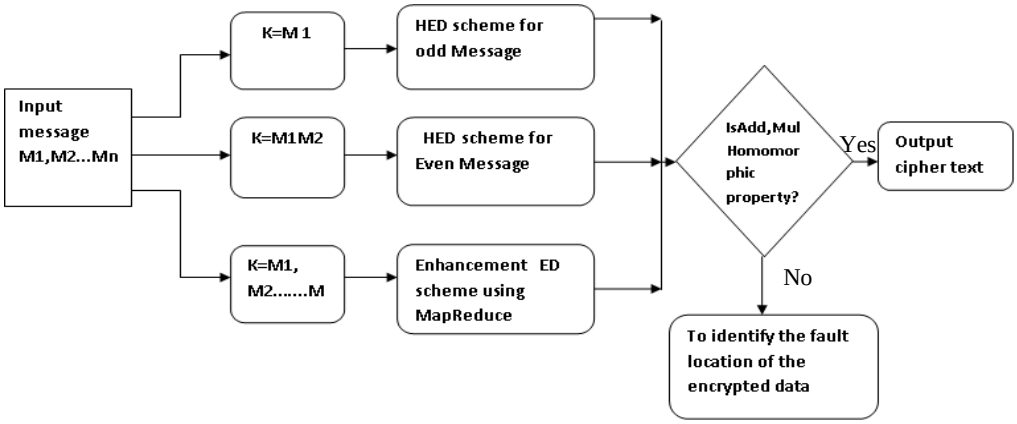

Figure 1. Homomorphic Error Detection Scheme

4.1. Multiplicative Property

Basic ED scheme select two successive chunks and perform encryption of two chunks. It calculates the multiplication of two chunks and perform the encryption the multiplication of two chunks. Basic ED scheme select two successive chunks and perform encryption of two chunks. Then calculate the multiplication of two chunks then perform the encryption the

multiplication. The chunk count to be processed is odd to generate constant chunk and add this chunk to input chunk list. The encryption process is initiated by encryption of two input chunks and buffers the corresponding results. The product of two chunks is calculated continued by performing encryption of the calculated chunks product. All the three schemes are decided to check the homomorphic property. When homomorphic property is satisfied it indicates no mismatch between the product of ciphertexts and the ciphertext of the product of chunks. When the homomorphic property not satisfied, it indicates fault on the encrypted data. Figure 2 shows homomorphic property of RSA.

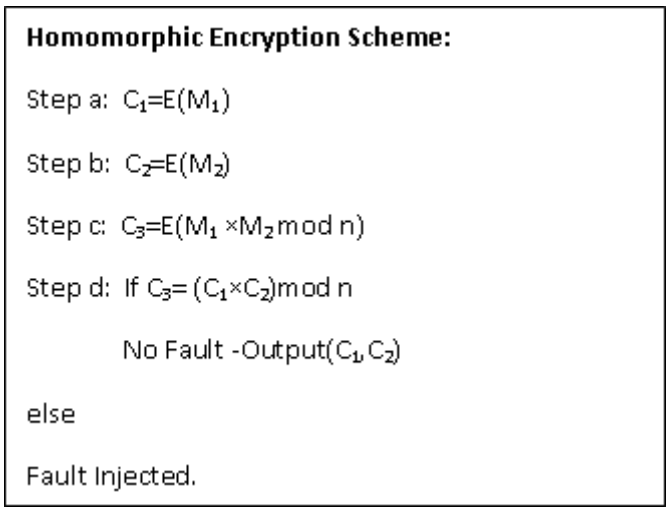

Figure 2. Homomorphic Property of RSA

4.2. Additive Property

Basic ED scheme select two successive chunks and perform encryption of two chunks. It calculates the addition of two chunks and perform the encryption the addition of two chunks. The chunk count to be processed is odd to generate constant chunk and add this chunk to input chunk list. The encryption process is initiated by encryption of two input chunks and buffers the corresponding results. The addition of two chunks is calculated by performing encryption of the calculated chunks addition. All the three schemes are decided to check the homomorphic property. When homomorphic propery is satisfied it indicates no mismatch between the addition of ciphertexts and the ciphertext of the addition of chunks. Figure 3 indicates the homomorphic property of pailler cryptosystem.

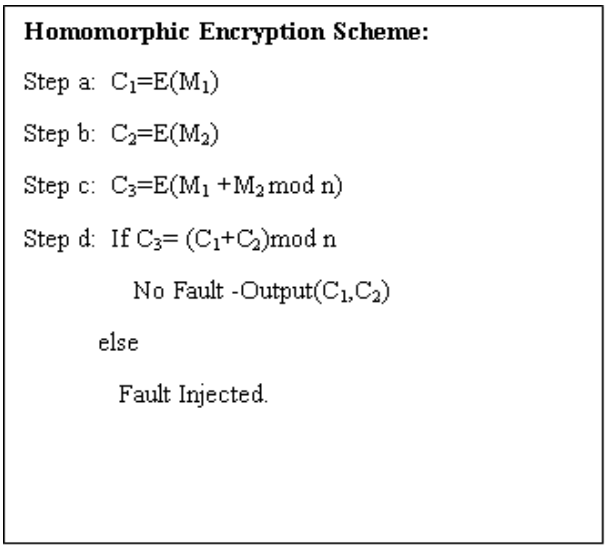

Figure 3. Homomorphic Property of Pailler Cryptosystem

4.3. ED Scheme for Even Chunks

The homomorphic error detection scheme operates on encrypted data in three steps namely one verification operation after three normal operations. The process starts with the encryption of two input messages and stores the result. The next step involves the encryption of the product of the two messages which occupies buffer. The comparison of the stored results is achieved through a comparator. The mismatch between the product of the ciphertext (i.e., $C_1 \times C_2$) and the ciphertext of the product of the chunks (C_3) shows an injection has occurred. Figure 2 represents the basic multiplicative homomorphic property supported by RSA.

On the otherside, during such operation fault can even occur inside comparator. Such problem can be resolved using a self-checking comparator or a duplicate running in parallel. The results are stored in registers along with the secret information such as p1 and q1. This prevents the attacker to steal the ciphertext before verification is done. The drawback in the scheme is that it requires always even number of chunks. Figure 3 gives a detailed design of error detection scheme for even chunks.

The homomorphic error detection scheme operates on encrypted data in three steps namely one verification operation after three normal operations. The process starts with the encryption of two input messages and stores the result. The next step involves the encryption of the addition of the two messages which occupies buffer. The comparison of the stored results is achieved through a comparator. The mismatch between the addition of the ciphertext (i.e., $C_1 + C_2$) and the ciphertext of the addition of the chunks (C_3) shows an injection has occurred. Figure 4.3 represents the basic additive homomorphic property supported by Paillier. On the otherside, during such operation fault can even occur inside comparator. Such problem can be resolved using a self-checking comparator or a duplicate running in parallel. The results are stored in registers along with the secret information such as p1 and q1. This prevents the attacker to steal the ciphertext before verification is done. The drawback in the scheme is that it requires always even number of chunks. Figure 4 gives a detailed design of error detection scheme for even chunks.

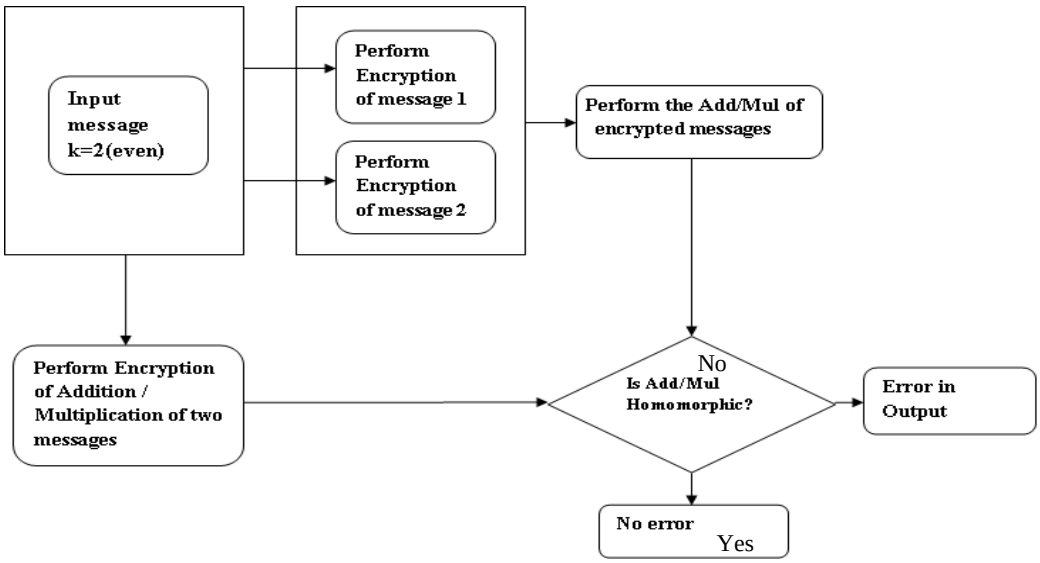

Figure 4. Error Detection Scheme for Even chunks

4.4. ED Scheme for ODD Chunks

The basic ED scheme supports even chunks. Therefore to support odd chunks the next scheme is proposed. Here introduction of padding solves the problem. In such cases the simple chunk M_1 has to follow the previous algorithm proposed with a constant chunk M_2. The chunk M_2 can either be a constant chunk or a random generated value. The scheme encrypts input chunk M_1. It is followed by the encryption of chunk $M_1 \times M_{cons}$ mod n for RSA and encryption of chunk $M_1 + M_{cons}$ mod n for Pailler cryptosystem. The comparison of the result with the product and addition of encryption of M_1 and constant chunk M_{cons} helps to identify the fault. Figure 5 represents a detailed design of error detection scheme for odd chunks.

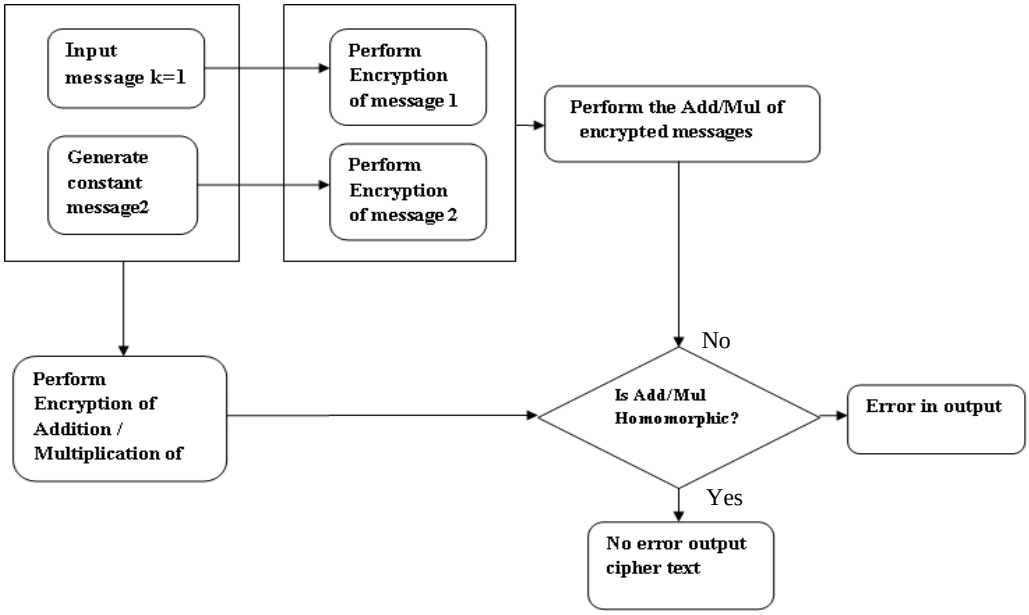

Figure 5. Error Detection Scheme for Odd Chunks

4.5. Enhanced ED Scheme

This scheme is designed for the chunks which are relatively large. The previous schemes time overhead ranges from 50% to 100%. To reduce the time overhead further, the third scheme is proposed by performing the same multiplicative and additive operation using Map-Reduce concept explained in the previous section. In this parallelized scheme, each mapper holds n chunks where the chunks are split based on the available number of mappers to compute the encrypted form of the input chunks. Mapper i..Mapper k are designed to perform encryption on the chunks resulting in $E(M_i)..E(M_j)$ under Mapper i to $E(M_k)..E(M_p)$ under Mapper k. The leftover odd chunk follow the padding scheme of constant chunk as explained previously. Figure 6 represents the design of enhanced error detection scheme.

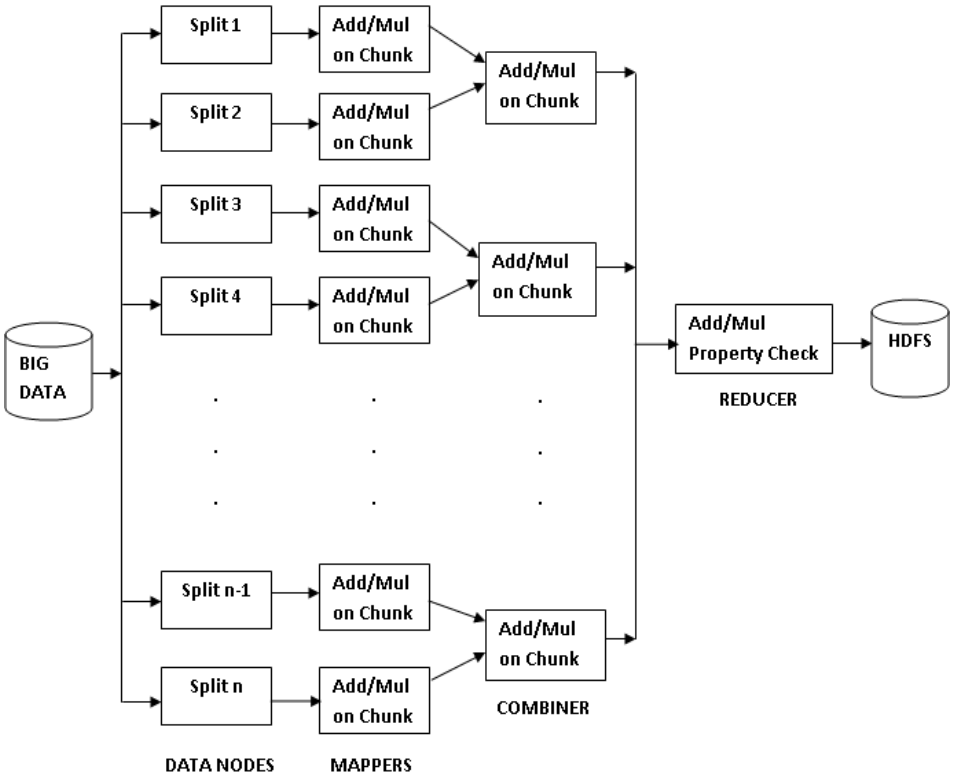

Figure 6. Enhanced ED scheme using map-reduce

6. Results and Discussions

6.1. Experimental Setup

The error detection scheme without Map-Reduce has an environment which has an Intel I5 processor running at 2.4 GHz, 4 GB RAM. The error detection scheme of RSA and pailler is implemented in Java. The scheme with Map-Reduce has an environment of 32 node cluster discussed in previous chapter. On examining the overall system performance, the proposed scheme efficiently detects fault attack on RSA and pailler system. Here the time overhead, hardware overhead and memory overhead are considered as performance metrics. Time overhead of proposed ED scheme varies according to the count of chunks and the number of nodes available for processing.

Basic ED scheme took two chunks for processing. It produces 50 % of the time overhead. The ED for odd chunks produces 100% of the time overhead. ED scheme using Map-Reduce produces minimum time overhead compared to the other two schemes.

6.2. ED Scheme for Even Chunks

The homomorphic evaluation of the basic scheme involves various metrics which are measured against memory, latency and running time. The memory overhead is due to the values of the buffers that hold the ciphertexts. While running the basic ED scheme the time overhead incurred includes computation of product of the chunks ($M_1 \times M_2$), encrypting the product $E(M_1 \times M_2 \bmod n)$, computation of the product of the ciphertexts of the two chunks ($C_1 \times C_2$) and comparing the results. But in pailler scheme, the time overhead incurred includes computation of addition of the chunks (M_1+M_2), encrypting the addition $E(M_1+M_2 \bmod n)$, computation of the addition of the ciphertexts of the two chunks (C_1+C_2) and comparing the results. The time overhead due to addition of messages are negligible when compared to the evaluation of encryption of addition of chunks. It results in 50% time overhead in the case of basic error detection scheme. But the time overhead incurred due to multiplications is relatively high when compared to the RSA sytem. It results in 50% time overhead in the case of basic error detection scheme. Fault detection latency which is defined as the time delay between the time of fault occurrence and the time of its detection is high when the fault occurs on the first chunk. It is therefore equal to the time cost of the three encryptions (i.e., $E(M_1)$, $E(M_2)$ and $E(M_1 \times M_2 \bmod n)$ for RSA and cost incurred for $E(M_1)$, $E(M_2)$ and $E(M_1+M_2 \bmod n)$ for Pailler. The output latency of both the schemes are high since the ciphertext comes out only after the verification.

While running the basic ED scheme fault detection latency which is defined as the time delay between the time of fault occurrence and the time of its detection is high when the fault occurs on the first chunk. It is therefore equal to the time cost of the three encryptions in both the cryptosystems (i.e. The output latency of the scheme is high) where the ciphertext comes out only after the verification.

6.3. ED Scheme for Odd Chunks

In the error detection scheme for odd number of chunks, the introduction of constant chunk has the advantages of holding the constant value in the buffer. It saves time of creating a random value whenever it is required by the comparator. Rather than generating the second chunk value and encrypting the buffer each and everytime it can hold it permanently and hence saves the overall running time. The cipher text of the constant chunk can be precomputed, Instead of generating randomly whenever required. The time overhead for encryption of product of single chunk is 100 percent. But such overhead due to the last chunk has little impact as the size of chunk increases. This methodology gives similar results in both the cryptosystems.

6.4 ED Scheme Using Map-Reduce

Due to parallelization, the time overhead incurred in computing encrypted form of each chunk is considerably reduced. The time taken to compute $c_1, c_2 .. c_n$ using Mappers relatively reduce the 50% overhead incurred before. However after computing the above ciphertext for n chunks the product of the chunks when calculated inside mapper_i ($M_i \times M_j$) and mapper_k

computing ($M_k \times M_p$) gives the result which when iterated yields the result of mapper$_i$ × mapper$_j$. Similarly Pailler cryptosystem yields addition of the chunks when calculated inside mapper$_i$ (M_i+M_j) and mapper$_k$ computing (M_k+M_p) gives the result which when iterated yields the result of mapper$_i$ + mapper$_j$. The time overhead due to multiplications and additions is negligible when running Map-Reduce on compute clusters with large number of nodes. This results in less than 20% time overhead which varies with the number of nodes of cluster. The memory overhead involves the usage of buffers to store the results of mappers. The worst case scenario of fault detection latency happens when the fault occurs on the first chunk. When compared to the other two schemes here the fault detection latency cost is reduced to two encryptions. (i.e., one-time encryption calculation of all chunks c_1.. c_n and $E(M_i \times .. \times M_k)$ for RSA and $E(M_i + .. + M_k)$. The output latency is relatively low even when it comes after verification as the parallel computation yields one-time calculation of encryption of chunks. When the homomorphic property not satisfied, it indicates fault on the encrypted data. Figure 7 gives an evaluation of RSA and Pailler for varied chunk sizes. It shows that Pailler consumes relatively more time than RSA. The enhanced scheme consumed relatively equal time for varied chunk sizes.

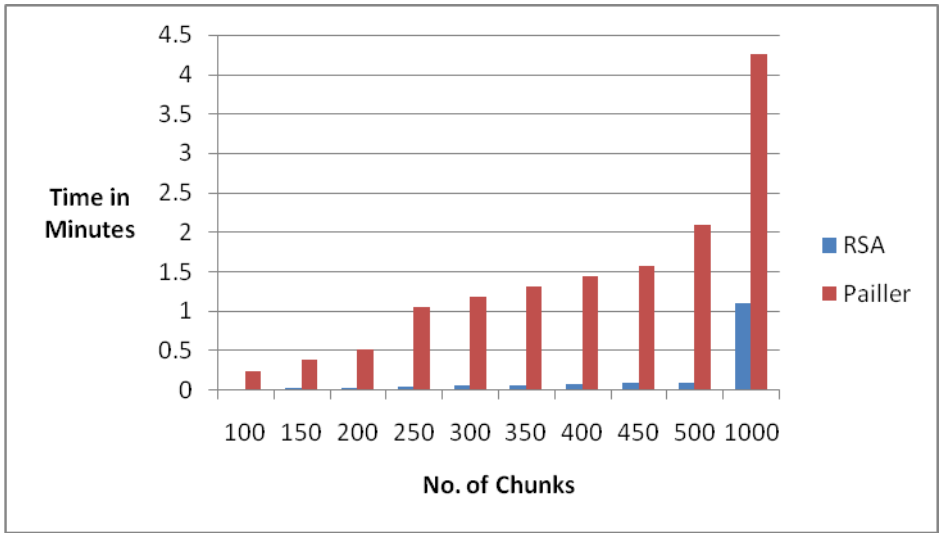

Figure 7. RSA Vs Pailler Cryptosystem

7. Conclusion

In this research, an effective low-cost and high-performance error detection scheme that uses additive homomorphic property of RSA and Paillier is introdued. The time overhead for the error detection scheme for odd and even chunks varied from 50% to 100%. The one-time encryption of individual chunks using Map-Reduce algorithm has significantly improved the fault detection latency in both RSA and Pailler cryptosystem. The memory overhead depends on the mapper output values that are stored in buffers. All the schemes support for output latency is relatively low as the verification to find the fault occurs only after comparing the output of the mapper.

References

[1] C. Wang, N. Cao, K. Ren and W. Lou, "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data", IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 8, **(2012),** pp. 1467-1479.
[2] K. Ren, C. Wang and Q. Wang, "Security Challenges for the Public Cloud", IEEE Internet Computing, vol. 16, no. 1, **(2012),** pp. 69-73.
[3] A. C. Melchor, S. Fau, C. Fontaine, G. Gogniat and R. Sirdey, "Recent Advances in Homomorphic Encryption: A Possible Future for Signal Processing in the Encrypted Domain", IEEE Signal Processing Magazine, vol. 30, no. 2, **(2013),** pp. 108-117.
[4] R. Nithiavathy, "Data integrity and data dynamics with secure storage service in cloud", International Conference on Pattern Recognition Informatics and Mobile Engineering (PRIME), **(2013),** pp. 125- 130.
[5] Y. Xun, K. M. Golam, P. Russell and B. Elisa, "Single-Database Private Information Retrieval from Fully Homomorphic Encryption", IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 5, **(2013),** pp. 1125-1134.
[6] G. Xiang, B. Yu and P. Zhu, "A algorithm of fully homomorphic encryption", 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), **(2012),** pp. 2030-2033.
[7] A. Yu, A. V.Sathanur and V. Jandhyala, "A partial homomorphic encryption scheme for secure design automation on public clouds", 21st Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), **(2012)** pp. 177-180.
[8] J. Yu, P. Lu, Y. Zhu, G. Xue and M. Li, "Toward Secure Multikeyword Top-k Retrieval over Encrypted Cloud Data", vol. 10, no. 4, **(2013),** pp. 239-250.
[9] N. Yukun, T. Xiaobin, C. Shi, W. Haifeng, Y. Kai and B. Zhiyong, "A security privacy protection scheme for datacollection of smart meters based on homomorphic encryption", EUROCON, **(2013),** pp.1401- 1405.
[10] A. Peter, E. Tews and S. Katzenbeisser, "Efficiently Outsourcing Multiparty Computation Under Multiple Keys", IEEE Transactions on Information Forensics and Security, vol. 8, no. 12, **(2013),** pp. 2046-2058.
[11] N. Saputro and K. Akkaya, "Performance evaluation of Smart Grid data Aggregation via homomorphic encryption", Wireless Communications and Networking Conference (WCNC), **(2012),** pp. 2945- 2950.
[12] X. Chen and Q. Huang, "The data protection of Map-Reduce using homomorphic encryption", 4th IEEE International Conference on Software Engineering and Service Science (ICSESS), **(2013),** pp. 419-421.
[13] L. Chen, Z. Tong, W. Liu and C. Gao, "Non-interactive Exponential Homomorphic Encryption Algorithm", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), **(2012),** pp. 224-227.
[14] D. J. Guan, E. S. Chen-Yu and T. Zhuang, "Detect Zero by Using Symmetric Homomorphic Encryption", Eighth Asia Joint Conference on Information Security (Asia JCIS), **(2013)**, pp. 1-7.
[15] J. Li, S. Chen and D. Song 2012, "Security structure of cloud storage based on homomorphic encryption scheme", 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS), **(2012),** pp. 224-227.
[16] C. Gentry and S. Halevi, "Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits", 52nd Annual Symposium on Foundations of Computer Science (FOCS), **(2011),** pp.107- 109.
[17] P. Zhu and G. Xiang, "The Protection Methods for Mobile Code Based on Homomorphic Encryption and Data Confusion", Fifth International Conference on Management of e-Commerce and e-Government (ICMeCG), **(2011),** pp. 256-260.
[18] F. Jin, Y. Zhu and X. Luo, "Verifiable Fully Homomorphic Encryption scheme", 2nd International Conference on Consumer Electronics Communications and Networks (CECNet), **(2012),** pp. 743-746.
[19] Z. Erkin, T. Veugen, T. Toft and R. L. Lagendijk, "Generating Private Recommendations Efficiently Using Homomorphic Encryption and Data Packing", IEEE Transactions on Information Forensics and Security, vol. 7, no. 3, **(2012),** pp. 1053-1066.
[20] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key cryptosystems", Communications of the ACM 21, no. 2, **(1978),** pp. 120–126.
[21] W. Diffie and M. E. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory, vol. 22, no. 6, **(1987),** pp. 644-654.
[22] D. J. Guan, T. Chen-Yu and E. S. Zhuang, "Detect Zero by Using Symmetric Homomorphic Encryption", Eighth Asia Joint Conference on Information Security (Asia JCIS), **(2013),** pp. 1-7.
[23] J. Dean and S. Ghemawat, "Map-Reduce:Simplified data processing on large clusters Commun.", ACM, vol. 51, no. 1, **(2008),** pp. 107-113.
[24] M. Stonebraker, D. Abadi, A. Batkin, X. Chen and M. Cherniack, "C-Store: A Column Oriented DBMS", VLDB, **(2005),** pp. 553-564.